MODULE 9p - Applets - Trials C

The applet to be developed in this series of trials will exploit the mouse. The first version is a simple modification of AppletB but should be saved in AppletC.java

```
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Button;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.event.MouseListener;                 // new statement
import java.awt.event.MouseEvent;                    // new statement

public class AppletC extends Applet                  // new class name
 { public String s = "Greetings";

   private Button jack = new Button("Jack");
   private Button jill = new Button("Jill");

   public void init()
    { this.addMouseListener(new AppML());            // new statement
      this.add(this.jack);
      this.jack.addActionListener(new JackL());
      this.add(this.jill);
      this.jill.addActionListener(new JillL());
    }

   public void paint(Graphics g)
    { g.drawRect(15, 15, 270, 70);
      g.drawString(this.s, 100, 60);
    }

   class AppML implements MouseListener              // new class
    { public void mousePressed(MouseEvent e)
       { AppletC.this.s = "Mouse pressed in applet";
         AppletC.this.repaint();
       }

      public void mouseReleased(MouseEvent e)
```

```
        { AppletC.this.s = "Mouse released in applet";
          AppletC.this.repaint();
        }

       public void mouseClicked(MouseEvent e)
        { AppletC.this.s = "Mouse clicked in applet";
          AppletC.this.repaint();
        }

       public void mouseEntered(MouseEvent e)
        { AppletC.this.s = "Mouse entered applet";
          AppletC.this.repaint();
        }

       public void mouseExited(MouseEvent e)
        { AppletC.this.s = "Mouse exited applet";
          AppletC.this.repaint();
        }
    }

   class JackL implements ActionListener
    { public void actionPerformed(ActionEvent e)
        { AppletC.this.s = "Jack pressed";            // new class name
          AppletC.this.repaint();                     // new class name
        }
    }

   class JillL implements ActionListener
    { public void actionPerformed(ActionEvent e)
        { AppletC.this.s = "Jill pressed";            // new class name
          AppletC.this.repaint();                     // new class name
        }
    }
 }
```

This program makes use of another method inherited from class Component:

```
addMouseListener()   -  used for adding an object of type MouseListener to
                        the putative listener array.
```

Like ActionListener, MouseListener is an interface (not a class) but it specifies that any class which implements the interface must incorporate FIVE methods; these deal with different usages of the mouse.

In this version of AppletC, class AppML is set up as the MouseListener class and this.addMouseListener(new AppML()) is used to add an instantiation of class AppML to the applet's listener array.

NOTES ON THE PROGRAM

To keep the program reasonably short, the println() statements have been removed as have the overriding methods start() stop() and destroy()

Java's understanding of a mouse is a little restricted. A mouse is deemed to have only one button (the left-hand button) and this button can be pressed, released or clicked (this last is a press followed by a release without moving the mouse between the two actions). Additionally a mouse can be detected entering or leaving a component (in this case the applet).

Together, these amount to five usages and for each usage there has to be a separate method in any class that implements the MouseListener interface.

In the present program each of the five methods simply sets String s to an appropriate message explaining which usage has been detected.

Notice that the MouseListener has been added to the applet's listener array and not to either button's listener array.

Any usage of the mouse provokes a MouseEvent and an instance of this Java class is handed to the relevant method in the MouseListener. Both class MouseListener and class MouseEvent need to be imported from the package java.awt.event

One may note a separate kind of listener, a MouseMotionListener, which is used for detecting mouse movement and mouse drag; this will be exploited later.

EXPERIMENTS WITH THE NEW APPLET

Compile the program using the javac command:

```
$ javac AppletC.java
```

The associated HTML needs to refer to AppletC.class so modify the old AppletB.html to AppletC.html as in:

```
<HTML>
  <BODY>
    <APPLET code="AppletC.class" width=300 height=100>
            Java is not available.
    </APPLET>
  </BODY>
</HTML>
```

Give the following appletviewer command:

```
$ appletviewer AppletC.html &
```

When the applet appears, try out the five usages of the mouse:

1. Press the left-hand button in the central area and HOLD IT DOWN.
   Mouse pressed in applet should appear.

2. Move the mouse before releasing the left-hand button and...
   Mouse released in applet should appear.

3. Repeat (1) and (2) but without moving the mouse at step (2).
   Mouse clicked in applet should appear.

4. Move the mouse outside the applet to provoke Mouse exited...

5. Move the mouse back inside the applet to provoke Mouse entered...

The Jack and Jill buttons should work normally.

THE MouseAdapter CLASS

Suppose you just want to detect the mouse entering the applet, then you still have to supply all five methods in class AppML even though four of them will have method headings but empty bodies.

As a somewhat strange facility, Java supplies a special class MouseAdapter which implements the MouseListener interface by providing all five methods with empty bodies:

```
   class MouseAdapter implements MouseListener
    { public void mousePressed(MouseEvent e) {}
      public void mouseReleased(MouseEvent e) {}
      public void mouseClicked(MouseEvent e) {}
      public void mouseEntered(MouseEvent e) {}
      public void mouseExited(MouseEvent e) {}
    }
```

At first sight, class MouseAdapter appears utterly useless! The reason for its existence is solely to save typing headings for methods which have no bodies. Given the availability of class MouseAdapter, you can write a MouseListener, such as AppML, by declaring that AppML extends MouseAdapter and supplying, as overrides, only the methods you want to use.

Such use of class MouseAdapter is exploited in the next version of the program...

A FIRST VARIATION

Modify the source code in AppletC.java so that it appears thus:

```
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Button;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.event.MouseAdapter;                  // modified statement
import java.awt.event.MouseEvent;

public class AppletC extends Applet
 { public String s = "Greetings";

   private Button jack = new Button("Jack");
   private Button jill = new Button("Jill");

   public void init()
    { this.addMouseListener(new AppML());
      this.add(this.jack);
      this.jack.addActionListener(new JackL());
      this.add(this.jill);
      this.jill.addActionListener(new JillL());
    }

   public void paint(Graphics g)
    { g.drawRect(15, 15, 270, 70);
      g.drawString(this.s, 100, 60);
    }

   class AppML extends MouseAdapter                  // modified class heading
    { public void mouseEntered(MouseEvent e)         // just one method in body
       { AppletC.this.s = "Mouse entered applet";
         AppletC.this.repaint();
       }
    }

   class JackL implements ActionListener
    { public void actionPerformed(ActionEvent e)
       { AppletC.this.s = "Jack pressed";
         AppletC.this.repaint();
       }
    }

   class JillL implements ActionListener
    { public void actionPerformed(ActionEvent e)
       { AppletC.this.s = "Jill pressed";
         AppletC.this.repaint();
       }
    }
 }
```

Compile the program using the javac command:

```
$ javac AppletC.java
```

Run the appletviewer program on this new applet:

```
$ appletviewer AppletC.html &
```

EXPERIMENTS WITH THE NEW APPLET

Unsurprisingly, the new applet has much less functionality than the previous version. When the mouse is moved into the applet the message Mouse entered applet appears but this won't now change until one of the Jack or Jill buttons is pressed.

A SECOND VARIATION

Modify the source code in AppletC.java so that it appears thus:

```
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Button;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;

public class AppletC extends Applet
 { public String s = "Greetings";

   private Button jack = new Button("Jack");
   private Button jill = new Button("Jill");

   public void init()
    { this.addMouseListener(new AppML());
                                                     // blank line
      this.add(this.jack);
      this.jack.addActionListener(new JackL());
      this.jack.addMouseListener(new JackML());      // new statement
                                                     // blank line
      this.add(this.jill);
      this.jill.addActionListener(new JillL());
    }
   public void paint(Graphics g)
    { g.drawRect(15, 15, 270, 70);
      g.drawString(this.s, 100, 60);
    }

   class AppML extends MouseAdapter
    { public void mouseEntered(MouseEvent e)
       { AppletC.this.s = "Mouse entered applet";
         AppletC.this.repaint();
       }
    }

   class JackL implements ActionListener
    { public void actionPerformed(ActionEvent e)
       { AppletC.this.s = "Jack pressed";
         AppletC.this.repaint();
       }
    }

   class JackML extends MouseAdapter                 // new class
    { public void mouseEntered(MouseEvent e)
       { AppletC.this.s = "Mouse entered Jack";
         AppletC.this.repaint();
       }
    }

   class JillL implements ActionListener
    { public void actionPerformed(ActionEvent e)
       { AppletC.this.s = "Jill pressed";
         AppletC.this.repaint();
       }
    }
 }
```

The principal modification is that a second MouseListener has been added by the statement:

```
                  this.jack.addMouseListener(new JackML());
```

The prefix this.jack rather than plain this indicates that this MouseListener is added to the Button jack rather than to the applet. One can add several listeners to a typical component and, of course, jack already has an ActionListener. The new MouseListener is an instantiation of the new class JackML.

Compile using javac and run the appletviewer program again.

Move the mouse from outside the applet to inside and then move it into Button jack. The message changes from Mouse entered Jack to Mouse entered Applet.

A similar MouseListener can be added to Button jill and any MouseListener can be used to detect the other forms of mouse usage either by overriding the appropriate methods of MouseAdapter or, perhaps, by supplying all five methods explicitly as in the first version of AppletC.

A THIRD VARIATION

Modify the source code in AppletC.java so that it appears thus:

```
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Button;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.event.MouseMotionListener;             // new statement
import java.awt.event.MouseMotionAdapter;              // new statement
import java.awt.event.MouseEvent;

public class AppletC extends Applet
 { public String s = "Greetings";

   private Button jack = new Button("Jack");

   public void init()
    { this.addMouseMotionListener(new AppMML());       // modified statement

      this.add(this.jack);
      this.jack.addActionListener(new JackL());
      this.jack.addMouseMotionListener(new JackMML()); // modified statement
    }
   public void paint(Graphics g)
    { g.drawRect(15, 15, 270, 70);
      g.drawString(this.s, 100, 60);
    }

   class AppMML implements MouseMotionListener         // modified class
    { public void mouseMoved(MouseEvent e)
        { AppletC.this.s = "Mouse moved in applet";
          AppletC.this.repaint();
        }

      public void mouseDragged(MouseEvent e)
        { AppletC.this.s = "Mouse dragged in applet";
          AppletC.this.repaint();
        }
    }

   class JackL implements ActionListener
    { public void actionPerformed(ActionEvent e)
        { AppletC.this.s = "Jack pressed";
          AppletC.this.repaint();
        }
    }

   class JackMML extends MouseMotionAdapter            // modified class
    { public void mouseMoved(MouseEvent e)
        { AppletC.this.s = "Mouse moved in Jack";
          AppletC.this.repaint();
        }
    }
 }
```

The principal modifications are that a MouseMotionListener has been added to the applet by the statement:

```
                  this.addMouseMotionListener(new AppMML());
```

and another has been added to Button jack by the statement:

```
                  this.jack.addMouseMotionListener(new JackMML());
```

Like MouseListener, MouseMotionListener is an interface but it specifies only two methods, mouseMoved() and mouseDragged(), each of which takes a MouseEvent argument. The first MouseMotionListener, AppMML, implements the interface by providing both methods.

Equivalent to class MouseAdapter there is class MouseMotionAdapter which provides both the required methods. The second MouseMotionListener, JackMML, extends this class and overrides the mouseMoved() method but leaves the mouseDragged() method with an empty body as inherited from MouseMotionAdapter.

Compile using javac and run the appletviewer program again.

When the mouse is moved or dragged inside the applet a flickering message appears in the central region. Both moving and dragging are continuous processes and the flicker rate indicates the frequency at which the events are being generated.

When the mouse is moved inside Button jack a flickering message appears in the central region announcing Mouse moved in Jack

A FOURTH VARIATION

Modify the source code in AppletC.java so that it appears thus:

```
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Polygon;                                    // new statement
import java.awt.event.MouseAdapter;
import java.awt.event.MouseMotionAdapter;
import java.awt.event.MouseEvent;

public class AppletC extends Applet
 { public Polygon poly = null;                              // new statement

   public void init()
    { this.addMouseListener(new AppML());                   // new statement
      this.addMouseMotionListener(new AppMML());
    }

   public void paint(Graphics g)                            // new paint() method
    { if (this.poly != null)
```

```
          g.drawPolygon(this.poly);
     }

    class AppML extends MouseAdapter                      // new class
     { public void mousePressed(MouseEvent e)
        { AppletC.this.poly = new Polygon();
          AppletC.this.poly.addPoint(e.getX(), e.getY());
          AppletC.this.repaint();
        }
     }

    class AppMML extends MouseMotionAdapter               // new class
     { public void mouseDragged(MouseEvent e)
        { AppletC.this.poly.addPoint(e.getX(), e.getY());
          AppletC.this.repaint();
        }
     }
 }
```

In this final version of AppletC both buttons have been dispensed with and the applet has been equipped with two listeners, a MouseListener AppML and a MouseMotionListener AppMML. The MouseListener AppML extends class MouseAdapter and overrides only the mousePressed() method. The MouseMotionListener AppMML extends class MouseMotionAdapter and overrides only the mouseDragged() method.

The program introduces a new class Polygon (which is imported from the java.awt package). A variable poly of type Polygon (potentially) stores a collection of point-pairs and these can be drawn on the central region as a connected set of points by the method g.drawPolygon().

Initially poly is null and the central region is blank. As soon as the mouse button is pressed the mousePressed() method in AppML is invoked. A new Polygon is assigned to poly and the first point is added by the addPoint() method. This takes two arguments representing the coordinates of the point to be added. These points can be extracted from the event argument, e, by e.getX() and e.getY().

If the button is held down and the mouse dragged around the central region the mouseDragged() method in AppMML is invoked; this happens at frequent intervals during dragging. At each activation a new point is added to poly and the call of repaint() ensures that the latest version of the polygon appears in the central region.

When the button is released, the latest polygon can be admired without any distracting flicker.

When the button is pressed again, the mousePressed() method in AppML is invoked again and assigns a new Polygon to poly. The old version becomes garbage and its image is cleared away at the call of repaint() in the mousePressed() method.

Compile using javac and run the appletviewer program again.

Observe the effect of pressing the mouse button and dragging the mouse around. Notice what happens if the mouse is dragged outside the applet and then back inside again. Check what happens when the button is released and then pressed again.